existing or new technology or data format can be integrated with metadata aware components once the metadata has been developed for the new structure. This means the development of a new type of component instantly makes the new functionality and data in a component available for use by any new or existing systems.

[0102] The component integration engine of the present invention may use command architecture that uses metadata to describe input, instructions, and output. Instructions are used to set parameters used by a command to determine how to perform the command. For example a command may take the name of an email transport protocol to determine how to communicate with an email server. Instructions can be described by metadata. Metadata can describe attribute names and value constraints, necessary method invocations, necessary constructors, and events that are generated during execution. A command provides metadata to describe the structure and type of the data it accepts as input. Metadata can describe attribute names and value constraints, available method invocations, available constructors, and events that are generated during execution of methods on the input data. A command provides metadata to describe the structure and type of data it produces as output. Metadata can describe attribute names and value constraints, available method invocations, available constructors, and events that are generated during execution of methods on the output data.

[0103] Commands may be combined to perform complex actions and create component integration. This integration of managed components, unmanaged components, structured data, and unstructured data through the "command architecture" can be used to produce applications without programming new code. Existing components provide access to resources, existing commands provide the functionality necessary, and metadata structures them together to produce the desired process.

[0104] If a data structure does not exist which is found necessary to perform a new process, the data structure may be created as a dynamic data structure, such as XML, or it may be compiled from source code into an object which is then made available to the run-time environment. In the present invention, a mechanism also exists to create a data structure using virtual implementations to create the structure and related operations.

[0105] If a function does not exist which is found necessary to create a new process, the function may be created dynamically by combining existing functions, or it may be compiled from source code into a new command which is then made available to the run-time environment.

[0106] In the present invention, a mechanism exists to request the performance of a command or a combination of commands. The request may be received from a variety of sources including but not limited to a network connection, a web page submission, or a GUI client.

[0107] A metadata-based XML marshalling/un-marshalling system allows the component integration engine to convert any object to XML by retrieving all of its attributes via its metadata, then writing the type of object and the name and value of each attribute as XML. This allows a component integration engine to automatically convert any object to and from an XML metadata format. The component integration engine can also use XSLT templates to convert from unknown XML formats to the expected metadata formats in order to allow it to construct appropriate objects for any XML request. XSLT commands exist in the command architecture. The component integration engine can use structured metadata customizers to access XML in structured formats other than the standard metadata format to convert XML to objects and objects to XML.

[0108] In some embodiments, the meta-implementation layer and component integration engine may include a Hierarchical Model View Controller that uses events based on metadata (an HMVC pattern). The model-view-controller pattern has been redesigned to use a "model"—"model controller"—"view controller"—"view" pattern, each part of which is allowed to be a hierarchy of objects. Communications occur only between adjacent parts. Communications between the model controller and view controller only occur at the top-most level of the hierarchy, instead of the traditional MVC or HMVC patterns which allow communication between any parts in the system and at any level. Adding this restriction between the model and view controllers allows greater distribution to occur by inserting a "forward" between them (leading to the pattern of "model"—"model controller"—"server-side forward"—"client-side forward"—"view controller"—"view"). Events define the values they carry using a definition. Upon initial registration of an event listener, the metadata definitions are passed to the listener followed by the current values. This innovation allows listeners to more correctly respond to events by adjusting to differences in the metadata, or use the metadata to more fully constrain values.

[0109] Metadata used in the meta-implementation layer and component integration engine of the present invention does not require that the object implement a specific interface, extend a specific parent model, or follow a specific naming convention as is required by other contemporary metadata mechanisms. Instead metadata is compiled into a customizer class that recognizes the metadata for a specific type of object. Compiled metadata is faster than dynamic metadata and can be applied to any object rather than objects that derive from a common class, implement a specific interface, or use a standard naming convention. The objects being described do not need to be designed or implemented differently in order to be described by metadata.

[0110] **FIG. 1** illustrates a meta-implementation layer of the present invention that includes a metametamodel repository, a metamodel repository and implementations. The metamodel repository includes enumeration descriptors, role descriptors, hint descriptors, datatype descriptors, constraint descriptors, attribute descriptors, other element descriptors, parameter descriptors, method descriptors, signal descriptors, interface descriptors, model descriptors, and package descriptors. The implementations include includes enumeration implementations, role descriptors, hint implementations, datatype implementations, constraint implementations, attribute implementations, other element implementations, parameter implementations, method implementations, signal implementations, interface implementations, model implementations, and package implementations.

[0111] The functioning of the various descriptors and implementations of the meta-implementation layer of **FIG. 1** are described in more detail below. Also, it should be